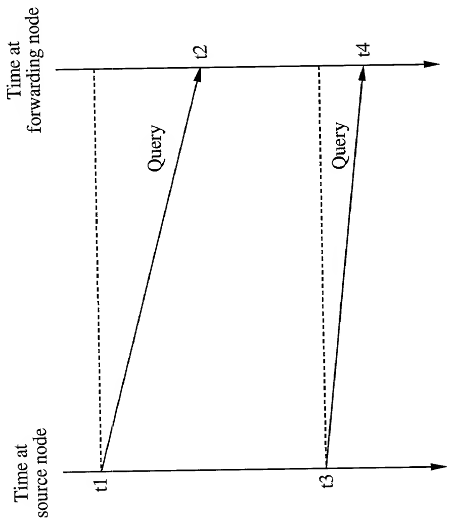
Time at forwarding node
t2
t4
Query
Query
Time at source node
t1
t3

FIG. 1

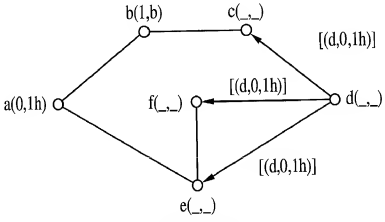

FIG. 2A

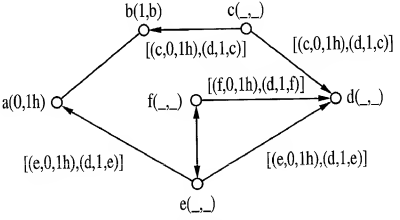

FIG. 2B

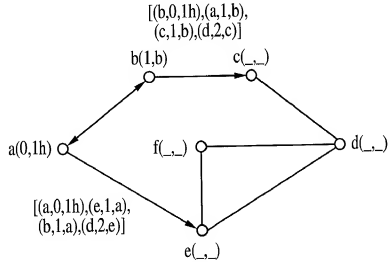

FIG. 2C

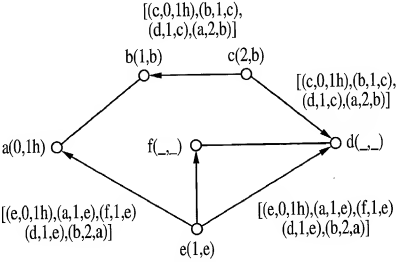

FIG. 2D

```
Procedure Init
called when node i initializes itself
begin
  N ← i
  D_i^i ← O
  s_i^i ← i
  p_i^i ← IP_LOCALHOST
  tag_i^i ← correct
  T_i^i ← present time
end

Procedure Recv_CU_Packet(pkt, nbr)
when node i receives a control packet from nbr
begin
  if (pkt.type = QRY)
    Query(pkt.nbr)
  else
    if (pkt.dst = BDCAST_ADDR)
      Update (pkt,nbr)
    else
      if(pkt.dst ∈ N and tag^i_pkt.dst = correct)
        Update (pkt.nbr)
  end else
end

Procedure Add_Dest(j)
called when node i learns of new destination j
begin
  N ← N ∪ j
  D_j^i ← ∞
  s_j^i ← NULL_ADDR
  p_j^i ← NULL_ADDR
  T_j^i ← present time
  for all (k ∈ N_i)
    D^i_jk ← ∞
    p^i_jk ← NULL_ADDR
  end for all
end
```

FIG. 3

```
Procedure Rmv_Dest( j)
called when node i removes j
begin
   N ← N - j
   for all (k ∈ N_i )
      remove j from k's array
   end for all
end

Procedure Add_Nbr(k)
called when node i learns of new neighbor k
begin
   N_i ← N_i ∪ k
   for all ( j ∈ N)
        D^i_jk ← ∞
        p^i_jk ← NULL_ADDR
   end for all
end

Procedure Rmv_Nbr(k)
called when node i learns of loss of neighbor k
begin
   N_i ← N_i - k
   for all ( j ∈ N)
      tag^i_j ← null
   send ← FALSE
   RT_update(send)
   If (send = TRUE)
      Send_Update(i, BDCAST_ADDR)
end
```

FIG. 4

Procedure DT_Update($k,j,RD_j^i rp_j^i$)
updating distance table entry
begin
  if ($RD_j^i < \infty$)
    $D_{jk}^i \leftarrow RD_j^i + 1$
  else $D_{jk}^i \leftarrow \infty$
  $p_{jk}^i \leftarrow rp_j^i$
  for all ($b \in N_i$)
    if k is in path from i to j via b
      $D_{jb}^i \leftarrow D_{kb}^i + RD_j^i$
  end for all
end

FIG. 5

```
Procedure Query(pkt,nbr)
called for processing query
begin
  for each entry (j, RD_j^i, rp_j^i) in pkt
    if (j ∉ N)
      if (RD_j^i = ∞)
        continue
      else
        Add_Dest(j)
        if (RD_j^i = 0)
          Add_Nbr(j)
      end else
    end if
    else
      if (RD_j^i = 0 and j ∉ N_i)
        Add_Nbr(j)
    end else
    DT_Update(pkt, src, j, RD_j^i, rp_j^i)
  end for each
  send ← F A L S E
  RT_Update(send)
  if (tag_pkt.dst^i = correct)
    Send_Update(pkt.dst, pkt.src)
  else
    if (present time - qr_pkt.dst^i > query_recieve_timeout)
      if (pkt.hops > 1)
        Send_Query (pkt.dst, (pkt.hops - 1), pkt.src)
      if ( pkt.hops ≥ 1)
        qr_pkt.dst^i ← present time
    end if
  end else
end
```

FIG. 6

```
Procedure Update(pkt,nbr)
called for processing update
begin
  newpath ← F A L S E
  if (pkt.dst ≠ B D C A S T _A D D R)
    if (pkt.src ∉ N or tag^i_pkt.src ≠ correct)
      newpath ← T R U E
  for each entry (j, RD^i_j, rp^i_j) in pkt
    if (j ∉ N)
      if (RD^i_j = ∞)
        continue
      else
        Add_Dest(j)
        if (RD^i_j = 0)
          Add_Nbr(j)
      end else
    end if
  else
    if (RD^i_j = 0 and j ∉ N_i)
      Add_Nbr( j)
  end else
  DT_Update(pkt.src, j, RD^i_j, rp^i_j)
 end for each
 send ← F A L S E
 RT_Update(send)
 if ( pkt.dst = B D C A S T _A D D R)
   if (send = T R U E) then Send_Update(i, B D C A S T _A D D R)
 else
   if ( pkt.dst = i)
     if (send = T R U E) then Send_Update(i, B D C A S T_A D D R)
   else
     if (newpath = T R U E and (pkt.src ∈ N or tag^i_pkt.src ≠ correct))
       newpath ← F A L S E
     if ( tag^i_pkt.dst = correct and newpath = T R U E
       and pkt.src is not in the path to pkt.dst)
       Send_Update(pkt.src, pkt.dst)
   else
     if (send) then Send_Update(i, B D C A S T _A D D R)
 end else
end else
end
```

FIG. 7

```
Procedure RT_Update(send)
updating routing table entries
begin
  for all ( j ∈ N)
    if (j = i)
      continue
    DTMin ← Min {D^i_jb ∀b ∈ N_i}
    if (D^i_jsj = DTM in) then ns ← s^i_j
    else ns ← b| {b ∈ N_i and D^i_jb = DTMin}
    x ← j
    loop ← F A L S E
    for (m = 0; m < | N |; m ++)
      visited [m] ← N U L L _A D D R
    num_visited ← 0
    while ((D^i_xns = M in { D^i_xb ∀b ∈ N_i })
        and D^i_xns < ∞ and tag^i_x ← null and loop = F A L S E)
        m ← 0
        while (m < num_visited)
          if (visited [m] = x or x = i)
            loop ← T R U E
        end while
        x ← p^i_xns
    end while
    if (loop = F A L S E and (p^i_xns = IP_LOCALHOST or tag^i_x = correct))
      tag^i_j ← correct
    else
      tag^i_j ← error
    if (tag^i_j = correct)
      if (D^i_j < DTMin) then send ← T R U E
      D^i_j ← DTMin
      s^i_j ← ns
      if (D^i_j = 1) then p^i_j ← i
      else p^i_j ← p^i_j
    end if
    end
      if (D^i_j =∞) then send ← T R U E
      p^i_j ← N U L L _A D D R
      s^i_j ← N U L L _A D D R
      D^i_j ←∞
    end else
  end for all
end
```

FIG. 8

```
Procedure Send_Update(src, dst)
broadcasting update
begin
   for each entry e (j , D_j^i, p_j^i) in routing table
       LIST ← LIST + e
   sort all entries in LIST in ascending distance values
   add each entry in LIST to pkt
   pkt.dst ← dst
   pkt.src ← src
   pkt.type ← UPDATE
   broadcast pkt to all neighbors
end

Procedure Send_Query(dest, hops, src)
broadcasting query
begin
   for each entry e (j, D_j^i p_j^i) in routing table
      LIST ← LIST + e
   sort all entries in LIST in ascending distance values
   add each entry in LIST to pkt
   pkt.dst ← dest
   pkt.src ← src
   pkt.hops ← hops
   pkt.type ← QUERY
   broadcast pkt to all neighbors
end

Procedure Buffer_Timer_Callback()
called periodically when buffer timer expires begin
   send ← FALSE
   Check_Buffer(send)
   if (send = TRUE) then Send_Update(i, BDCAST_ADDR)
end
```

FIG. 9

Procedure Get_Route_For_Pkt(*dest*)
decides if query needs to be sent and sends it
begin
  If (((present time − qs^i_{dest}) > *query_send_timeout)*
    and ($hqs^i_j = MAX\_HOPS$))
    $hqs^i_j \leftarrow ZERO$
    $zqs^i_j \leftarrow$ present time
    Send_Query(dest, 0, i)
end if
If (((present time − qs^i_{dest}) > *query_send_timeout)*
    and ($hqs^i_j = ZERO$)
    and (present time − zqs^i_j) > zero_*qry_send_timeout)*
    $hqs^i_j \leftarrow MAX\_HOPS$
    $qs^i_j \leftarrow$ present time
    $qr^i_j \leftarrow$ present time
    Send_Query(*dest, MAX_HOPS, i*)
  end if
end if

FIG. 10

```
Procedure Handle_Data_Pkt (pkt,nbr)
data packet can be from an upper layer or a forwarded pkt from nbr
begin
   If ( pkt.dst = i)
      send packet to correct upper layer port
   else if ( pkt.src = i)
      j ← pkt.dst
      if ( j ∈ N and tag_j^i = correct)
         send pkt to s_j^i
      else
         queue pkt in buffer
         Get_Route_For_Pkt ( pkt.dst)
      end else
    end else if
    else
       j ← pkt.dst
       if( j ∈ N)
       if (nbr = s_j^i)
          Send_Update(i, BDCAST_ADDR)
          drop pkt and return
       end if
       if (tag_j^i=correct)
          send pkt to s_j^i
    else
          Send_Update(i, BDCAST_ADDR)
          drop pkt
    end if
    else
       queue pkt in buffer
       Get_Route_For_Pkt ( pkt.dst)
    end else
  end else
end
```

FIG. 11

```
Procedure Check_Buffer(send)
checks buffer to forward or drop packets
begin
   for each  pkt in buffer
    if (time pkt has been in buffer > data_pkt_timeout)
        drop pkt
        return
    end if
    j ← pkt.dst
    if ( pkt.src=i)
        if ( j ∈ N and tag_j^i = correct)
            send pkt to s_j^i
    else
            Get_Route_For_Pkt(pkt.dst)
    end if
    else
    if ( j ∈ N)
        if( tag_j^i = correct)
            send pkt to s_j^i
        else
            send ← TRUE
            drop pkt
          end else
         end if
        else
            Get_Route_For_Pkt(pkt.dst)
      end else
   end for each
end
```

FIG. 12

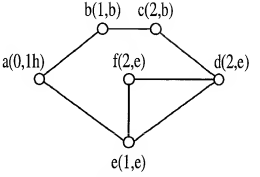

FIG. 13A

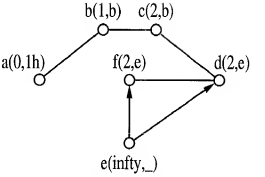

FIG. 13B

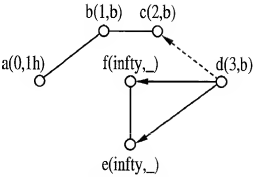

FIG. 13C

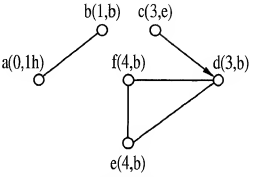

FIG. 13D

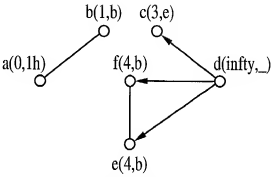

FIG. 13E

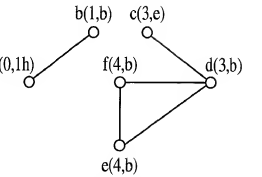

FIG. 13F

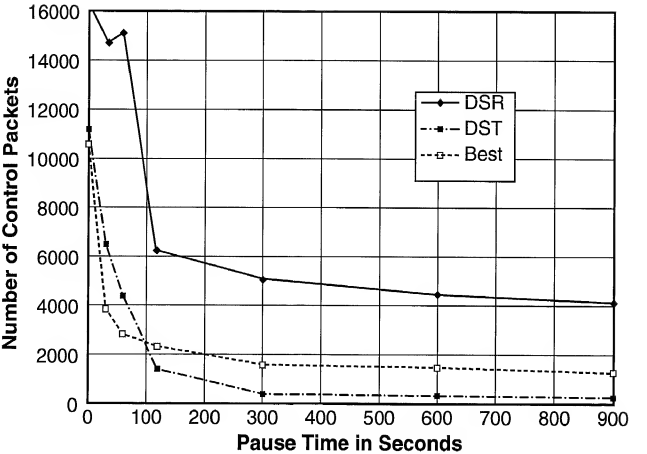
16000
14000
12000
10000
8000
6000
4000
2000
0
Number of Control Packets
0
100
200
300
400
500
600
700
800
900
Pause Time in Seconds
DSR
DST
Best

FIG. 14

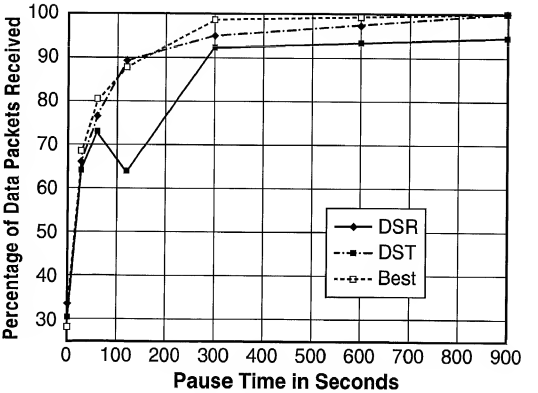

FIG. 15

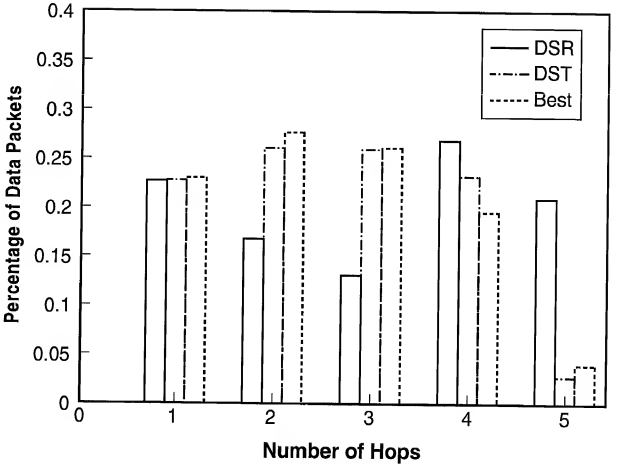

FIG. 16

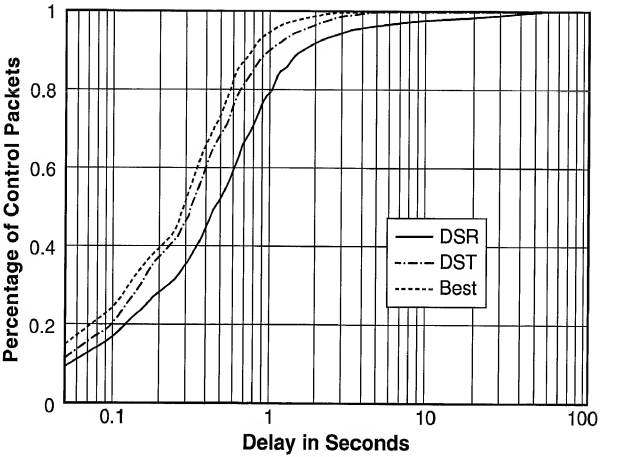

FIG. 17

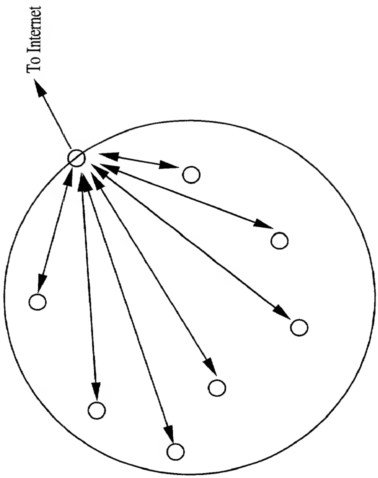

FIG. 18

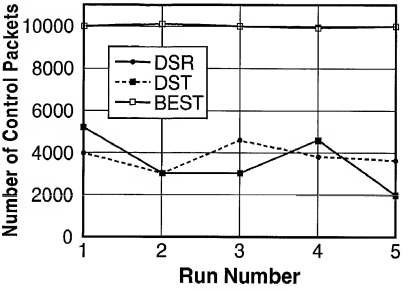

FIG. 19

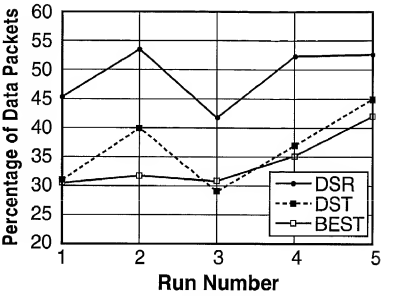

FIG. 20

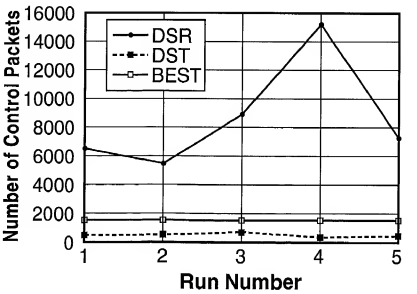

FIG. 21

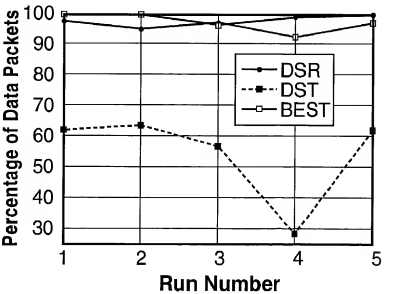

FIG. 22